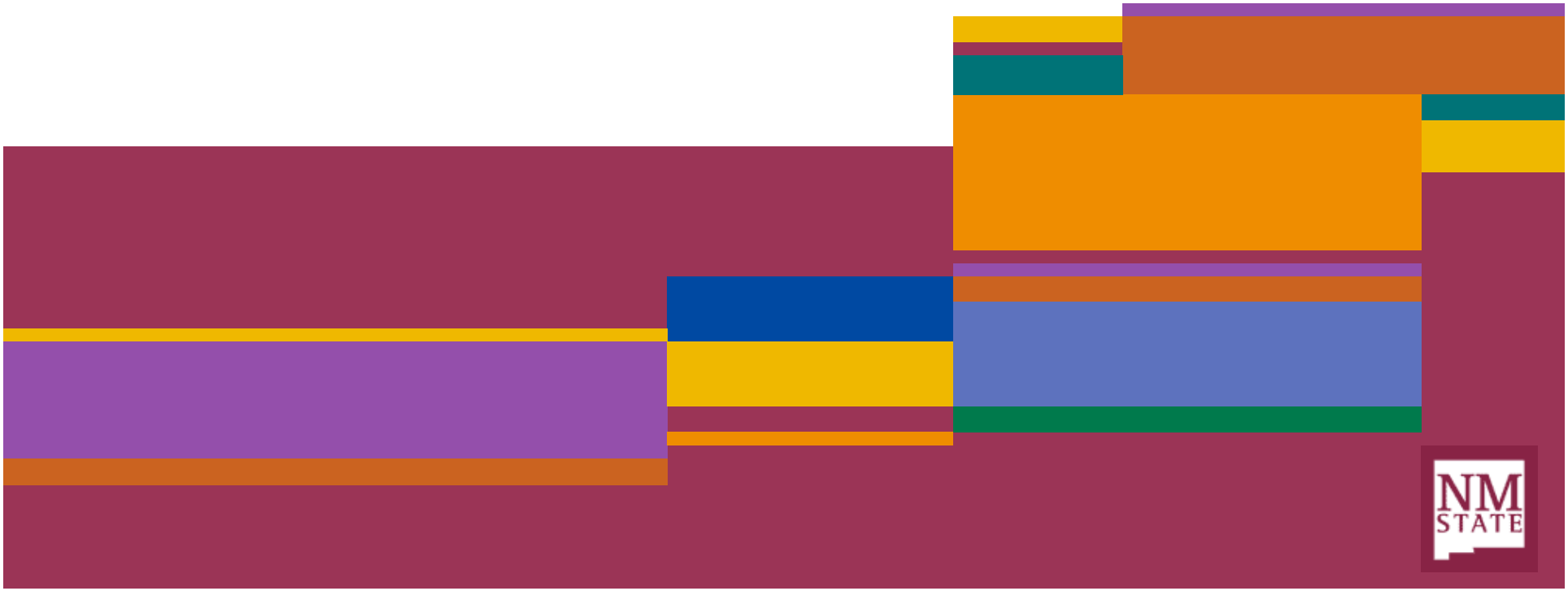
PortAudio Application Programming Interface (API) Part 1



References

- Material for this lecture is drawn from the PortAudio (PA) website:

  `http://www.portaudio.com/`

- The PA tutorial can be found at

  `http://www.portaudio.com/trac/wiki/TutorialDir/TutorialStart`

Introduction

- PortAudio is a free, open-source, cross platform, ‘C’ library for audio I/O
  - We can write a simple audio program in ‘C’ that will compile and run on Linux, Macintosh OS X, and Windows platforms
  - This means that you can write a single ‘C’ program to process an audio signal using a sound card for audio I/O, and that program can run on several different platforms just by recompiling the source code
- PortAudio provides a very simple API for accessing the soundcard using a simple “callback” function
  - Prof. De Leon has written a passcode to utilize the PortAudio API for developing real-time DSP applications using a soundcard

Introduction (cont.)

- Here are the steps to writing a PortAudio application:

1. Write a *callback* function that will be called by PortAudio when audio processing is needed
2. *Initialize* the PA library and *open* a stream for audio I/O
3. *Start* the stream. Your callback function will be now be called repeatedly by PA in the background
4. In your callback you can read/write audio data from/to inputBuffer/outputBuffer (and process in between)
5. *Stop* the stream by returning 1 from your callback, or by calling a stop function
6. *Close* the stream and *terminate* the library

Installing PortAudio

- To begin, you must first download PortAudio

`http://www.portaudio.com/download.html`

and build/install for your system. System-dependent directions are given at

`http://www.portaudio.com/trac/wiki/TutorialDir/TutorialStart`

1. Writing a Callback Function

- To write a program using PortAudio, you must include the “portaudio.h” header (include) file
  - You may wish to examine “portaudio.h” since it contains a complete description of the PortAudio functions and constants
- The next task is to write your own “callback” function
  - The callback is a function that is called by the PortAudio engine whenever it has received audio samples or when it needs to transmit audio samples

1. Writing a Callback Function (cont.)

- Before we begin, it's important to realize that the callback is a *delicate* place
  - Some systems perform the callback in a special thread, or interrupt handler, and it is rarely treated like the rest of your code therefore...

1. If you want your audio to reach the soundcard on time, you'll need to make sure whatever code you run in the callback runs *quickly*

2. As a rule of thumb, don't do anything like allocating or freeing memory, reading or writing files, printf(), or anything else that might take an unbounded amount of time or rely on the OS or require a context switch

3. Also do not call any PortAudio functions in the callback except for perhaps Pa_StreamTime() and Pa_GetCPULoad()

1. Writing a Callback Function (cont.)

- Your callback function must return an int and accept the exact parameters specified in this typedef:

```
typedef int PaStreamCallback(void *inputBuffer, void *outputBuffer,
unsigned long framePerBuffer, const PaStreamCallbackTimeInfo* timeInfo,
    PaStreamCallbackFlags statusFlags, void *userData);
```

- See De Leon's passcode lines 40 – 64 for implementation of the callback function
  - Note that in the example, samples are of type short (16 bit) but other types such as float can be used

2. Initializing PortAudio

- Before making any calls to PortAudio, you *must* call Pa_Initialize()
  - This triggers a scan of available soundcard which can be queried later
  - Like most PA functions, it will return a result of type paError. If the result is not paNoError, then an error has occurred, i.e.

```
err = Pa_Initialize();
if( err != paNoError ) goto error;
```

- You can get a text message that explains the error message by passing it to Pa_GetErrorText(err)

```
printf("PortAudio error: %s\n", Pa_GetErrorText(err));
```

2. Opening a Stream using Defaults

- The next step is to open a stream, which is similar to opening a file
  - You can specify whether you want audio input and/or output, how many channels, data format, sample rate, etc.
  - Opening a *default* stream means opening the default input and output devices, which saves you the trouble of getting a list of devices and choosing one from the list
- See Prof. De Leon's passcode lines 80 – 91 for implementation of the above
  - In De Leon's PortAudio passcode, we choose stereo (2 channels) I/O, 16 bit (short) samples, and 48000 Hz sample rate

3. Starting a Stream

- After calling Pa_StartStream(), PortAudio will start calling your callback function to perform the audio processing

```
err = Pa_StartStream( stream );
if( err != paNoError ) goto error;
```

  - You can communicate with your callback routine through the data structure (*inputBuffer, *outputBuffer) you passed in on the open call

- See Prof. De Leon's passcode lines 93 – 95 for implementation of the above

3. Starting a Stream (cont.)

- PortAudio will continue to call your callback function and process audio until you stop the stream. This can be done in one of several ways.
- One way is by sleeping (PA processes audio for a period of time and then sleeps).

```
/* Run for 60 seconds. */
Pa_Sleep(60*1000);
```

- See Prof. De Leon's passcode lines 97 – 103 for implementation of the above and an implementation which runs until the RETURN is hit

5. Stopping a Stream

- There are several ways to stop playback, the simplest is to call Pa_StopStream()

```
err = Pa_StopStream( stream );
if( err != paNoError ) goto error;
```

- Pa_StopStream() is designed to make sure that the buffers you've processed in your callback are all played out, which may cause a delay before stopping
- See De Leon's passcode lines 105 – 107 for implementation of the above

6. Closing a Stream and Terminating PortAudio

- When you are done with a stream, you should close it to free up resources:

```
err = Pa_CloseStream( stream );
if( err != paNoError ) goto error;
```

6. Terminating PortAudio (cont.)

- It is also important, when you are done with PortAudio, to terminate:

```
Pa_Terminate();
printf("Program completed.\n");
return err;
```

- See Prof. De Leon's passcode lines 113 – 115 for implementation of the above

Utility Functions

- In addition to the functions described elsewhere in this tutorial, PortAudio provides a number of utility functions which are useful in a variety of circumstances
- You can get the error message from an error number using Pa_GetErrorText:

```
const char * Pa_GetErrorText (PaError errorCode)
```

- See Prof. De Leon's passcode lines 117 – 122 for use and implementation of the above

DeLeon's PortAudio Passcode

- Review of complete PA passcode